

---

# Faster Constraint Solving Using Learning Based Abstractions

---

Sumanth Dathathri  
California Institute of Technology

Nikos Arechiga  
Toyota ITC

Sicun Gao  
MIT CSAIL

## 1 Introduction

This work addresses the problem of scalable constraint solving using satisfiability modulo theories (SMT) solvers. We propose a novel technique that combines traditional constraint-solving approaches with machine learning techniques to propose abstractions that simplify the problem. Our technique is complementary to existing constraint solving approaches, in the sense that it can be used to improve the scalability of an existing tool.

Given a large set of constraints, we propose heuristics to partition the set of constraints into smaller subsets that are well suited to individually being *abstracted* into simpler constraints. Subsequently, we formulate an asymmetric learning approach that learns simpler representations of these subsets. Once we have learnt a simpler representation of the original set of constraints, we solve these simpler overapproximating (or underapproximating) constraints to find solutions to the original constraints.

**Related Work** Theorem-proving techniques attempt to symbolically derive a solution for the set of constraints, or prove that none exists [4, 3]. These techniques, however, generally suffer from a lack of automation, and must be assisted by a trained human user. Deployment of theorem proving techniques for industrial applications would require extensive re-training of engineers and increase the time and cost of system development.

On the other hand, fully-automatic SMT solving techniques cannot yet be scaled to the requirements of industrial models. In [5] a SVM based algorithm is proposed to automatically learn invariants of programs, which can be used in a proof of program correctness. Their techniques could still suffer from scalability issues, and additionally are not well suited to learning simpler overapproximations for constraints with non-superficial non linearities.

## 2 Learning Abstractions

We use a *semi-soft* margin SVM, where we allow only samples of one type to be misclassified. Notice that this deviates from the standard soft-margin SVM in the sense that only one type of data-points are provided with slack-variables. In a set of data-points with labels in  $\{+1, -1\}$ , if none of the points labelled as  $+1$  are misclassified (by restricting the slack variables to only those that are labelled  $-1$ ) the classifier overapproximates the points labelled  $+1$ .

Given a set of constraints, they are decomposed using the heuristics proposed in Section 3. Following the decomposition, points satisfying and falsifying these smaller subsets of constraints are sampled using SMT solvers (z3[1] and dReal[2]). For these sampled points, a classifier is learnt in a higher dimension space (using kernels). Let  $A(x)$  be the subset of constraints and  $g(x)$  be the classifier learnt. Using a SMT solver we check that  $\forall x. A(x) \implies (g(x) > 0)$  holds.

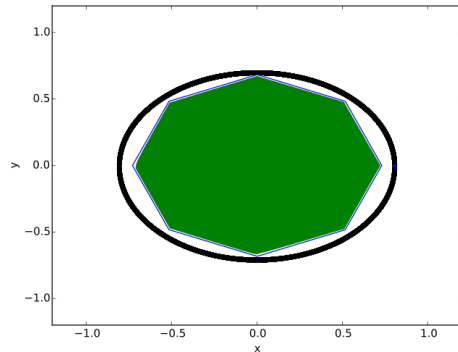


Figure 1: Quadratic classifier overapproximating the feasible region formed by the intersection of half-spaces

Since, the points satisfying the constraints are not misclassified, it is likely that the classifier learnt is an overapproximation. However if this fails, a counter example guided refinement loop (CEGAR) is used where counter examples to the condition are added to the training data-set and a new classifier is learnt. This is repeated until a classifier is found that successfully overapproximates the feasible set for constraints  $A(x)$ . A similar approach can be used to learn underapproximations.

### 3 Heuristics for Decomposition

Our current implementation uses the heuristics described below to decompose the original set of constraints into smaller subsets. We use these heuristics frequently in sequence.

**Hamming decomposition** This heuristic groups together clauses that have many common variables—i.e., those clauses whose sets of variables, treated as vectors, have a small Hamming distance. Clauses that have all common variables will have a distance of 0. We assume that a maximum distance bound  $\theta$  is given, such that any two clauses that have a Hamming distance less than or equal to  $\theta$  will be grouped together for abstraction.

**Sampling-based decomposition** In cases when the Hamming decomposition provides too few clause classes, we refine the classes by sampling the clauses and grouping together clauses that share many satisfying instances. Suppose a set of points  $Z$  that satisfy clause  $c_m$  is given, and suppose  $Z|_{c_n} \subseteq Z$  denotes the points of  $Z$  that satisfy clause  $c_n$ . Then, we define the *sample distance* with respect to  $Z$  as

$$S_Z(c_m, c_n) = 1 - \frac{\text{card}(Z|_{c_n})}{\text{card}(Z)}.$$

Clauses with a sampling distance less than a given  $\theta$  will be grouped together.

### 4 Faster constraint solving

The simpler consequents (overapproximations) learnt are solved using an SMT solver. In parallel, the antecedents (underapproximations) are similarly determined and are solved using an SMT solver. If the joint consequents are infeasible, then the original set of constraints are infeasible. If the joint antecedents have a solution, the solution also satisfies the original set of constraints. However, if the joint consequents have a solution and the joint antecedents have no solution, we cannot conclude anything about the original set of constraints.

### 5 Benchmark experiments

Our experiments on 20 randomly generated benchmarks show that our technique allows improved handling of constraint solving instances that are slow to complete on a conventional solver. For the vast majority of the benchmarks, solving the abstracted constraints in general takes an order of magnitude time lesser than solving the original problem directly on the solvers z3 and dReal. For simpler instances of constraint solving, the time for solving including abstraction is larger than directly solving the problem. However for larger complex instances that are slow to complete on z3 and dReal, the time for solving including the time for abstraction is comparable and at times, better. Some benchmarks that time-out on both z3 and dReal are simplified and solved using the abstraction based approach proposed here in reasonable time.

### References

- [1] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, TACAS’08/ETAPS’08*, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
- [2] Sicun Gao, Soonho Kong, and Edmund M. Clarke. *dReal: An SMT Solver for Nonlinear Theories over the Reals*, pages 208–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [3] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- [4] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.
- [5] R. Sharma, A. V. Nori, and A. Aiken. Interpolants as classifiers. In *Computer Aided Verification*, 2012.